平成28年度　修士論文

OpenFlowによる高信頼・トラヒック分散ネットワークの構築

所属　：　情報理工学研究科
情報・通信工学専攻
大木研究室
提出者　：　1431063 鈴木 龍太
主任指導教員：　大木英司 教授
指導教員：　來住直人 教授
提出日　：　2017年 1月 30日

概要

近年の高度情報化社会に於いてはネットワーク通信に対する需要が非常に高まっており、障害が発生した際の被害をなるべく回避することでネットワークの信頼性を高めるように努めることは、非常に重要であると言える。その一方で、破損時の対策だけでなく、平常時の通信に於ける効率的な通信もまた両立させたい。

ネットワークの効率性を向上させる一つの手段として、現在のネットワークに於いて使われている Open Shortest Path First(OSPF) というルーチングプロトコルを改良した Smart-OSPF (S-OSPF) が Mishra らによって提案されており、これは発ノード（出発点のノード）においてトラヒックを分散させて送ることにより、ネットワークの混雑を抑えた通信を可能とするという概念である。

S-OSPF には現在 2 つの課題がある。1 つは、計算されたトラフィック分散比をルータに通知するコントローラを用意し、且つルータにトラヒックを分散する機能を実装する為の改修をする必要がある。もう 1 つは、分散中にネットワーク故障が起こった時の対処方法が現在確立されていないので、その際の対策法を検討する必要がある。

これまでになされた S-OSPF を実際のネットワークへ実装させた場合及びその故障発生時の対応動作の性能へのアプローチは、主に CPLEX という数理計画問題を解く計算ソフトウェアを用いたシミュレーションによるもので、実際のネットワーク機器を用いての研究はまだされ始めてきたばかりの段階である。また、ネットワークをソフトウェアによって制御する Software-Defined Network（SDN）によって S-OSPF を実機へ実装する研究は既に行われているが、こちらは故障発生時の対策については未だ考慮である。

本研究では、ネットワーク機器を一つのコントローラで一元管理する OpenFlow によって、これらの従来研究を踏まえた効率的な平常時の通信及び経路に故障が発生した際における適切な対応動作の両立について検討する。

目 次

第1章　序論

1.1　研究背景・目的

インターネットやスマートフォンの普及、更にクラウドコンピューティングの登場等により、インターネット通信のトラヒックはここ近年でかなり増大してきている。今やインターネット通信は人々にとって様々な面で必要不可欠なインフラ要素の一つだ。しかしそれだけに、何らかの要因によって障害が発生した際の影響も甚大になることが予測される為、その被害を最小に留めるように努めることは、ネットワークの信頼性を高める為に非常に重要であると言える。通信障害発生時の迅速な通信復旧という面においてはNICT（情報通信機構）による震災時の通信インフラ復旧 [1] をはじめ、これまで様々なアプローチがなされてきた。しかし、そうした破損時の対策だけでなく、平常時の通信に於ける効率的な通信もまた同時に両立させたいと考えた。

ネットワークの効率性を向上させる一つの手段として、Open Shortest Path First(以下、OSPF) というルーチングプロトコルがある。ネットワークを構成しているノード同士が各々の経路情報を交換することで、最短経路を導出する。

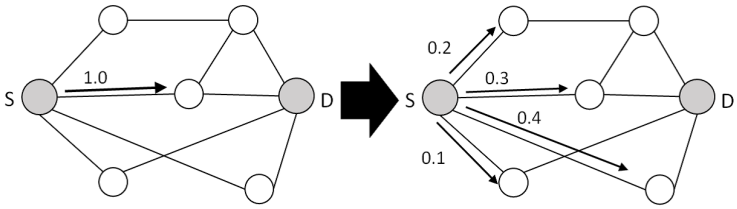

図 1.1: OSPF（左）と、S-OSPF（右）

このOPSFを改良したSmart-OSPF (以下, S-OSPF) がMishra らによって提案されている [2]。発ノード（出発点のノード）においてトラヒックを適切な比率で分散させて送ることにより、ネットワークの混雑を抑えることを目的とする。S-OSPFを実際のネットワークへ実装した場合のその通信輻輳を減らす性能については、これまでCPLEX[3] という数理計画問題を解く計算ソフトウェアを用いたシミュレーションによってアプローチされてきた [4]。

しかし, 現在S-OSPFを実装させるには二つの課題がある。一つはリンク故障が起こった時の対処が現在確立されておらず、障害発生時には通常の最短経路ルーチングに戻ってしまう可能性がある為、その際の復旧又は対策のメカニズムを検討しなくてはならない。もう一つはトラフィック分配比の計算には集中制御を必要

とする為、既存のルータに異なる分散比でのトラヒック転送をさせるには比率計算の為のサーバーを用意し、且つルータに分散機能を実装させる為の改修を施さなくてはならないが、ルータはその機能や規格がベンダー依存である為、ルータごとの改修に手間がかかる。

そこで、OpenFlow[5] という、ネットワークを構成しているスイッチを一つのコントローラで一元管理する技術がある。これは、ソフトウェアによって制御されるネットワーク Software-Defined Network（以下、SDN）に用いられているもので、この OpenFlow による S-OSPF の実装をする取り組みがこれまでになされてきた[6]。この技術を用いれば、ベンダーの定めた機器の規格や設定方法を気にすることなく、プログラミングによって機能をネットワークへ実装することができるからである。しかし、OpenFlow を用いて S-OSPF を実装するにあたってもまだ課題はある。まず OpenFlow 環境を構築するにはホストやスイッチ等のネットワーク機器を用意しなくてはならならず、またこれまでの OpenFlow による実装の研究は障害発生時の対策については未だ十分な考慮がされてない段階である。

本研究では実際にホストやスイッチを用意することなく、仮想的に OpenFlow を動かすことができる mininet[7] を用いることによって、S-OSPF の使用中に経路に障害が発生した際に於ける適切な対応動作を実現させることを目的とする。

更に、今回実装するルーチングは、実装難易度などの観点から、発ノードにおけるトラヒック分散を行わず、最も輻輳を回避できる経路を一つ選択し、そこへトラヒックを転送する non-slit S-OSPF[8] を採用した。これを用いて、ネットワーク全体の通信の分散を目指す。

1.2 論文構成

本論文では、まず第 2 章で S-OSPF と non-split S-OSPF 及び OpenFlo wの説明、第 3 章で mininet を用いた実験環境及びそのメカニズム、第 4 章で平常時の負荷分散と経路破損時の対応動作の実験及びそれらの結果、第 5 章でまとめを述べる。

第2章　従来技術と関連研究

2.1　S-OSPF

S-OSPF の課題

現在のネットワークにおいて最も主流として使われている OSPF というルーチングプロトコルがある。これは通信を行う際に、ネットワークを構成している各ノードがそれぞれの経路情報を互いに交換し、リンクコストを計算することで出発点から目的地までの最短経路を自動で算出し、その経路を選んで通信を行うというものである。しかし、このルーチングによってリンクコストのみを考慮して選択された経路は、必ずしも通信の輻輳回避という点では適切でない場合もある。

そこで、OSPF を改良した S-OSPF が提案されている。S-OSPF では発ノードでトラヒックを隣接ノードへ適切な比率で分散させて送る。中継ノードでは分散されたトラヒックを OSPF に従って着ノード（目的地のノード）まで送っていく。このようにしてトラヒックの輻輳を回避し、通信の効率化を図る。

S-OSPF はリンク故障発生時の対策が未確立な為、その際に起こりうる動作が想定できず、場合によってはネットワーク上に大きな輻輳を生み出してしまう可能性がある。信頼性の高いネットワークの構築を目指すのであれば、こうした影響をなるべく最小に留める為に、故障発生時の対策を検討しなくてはならない。

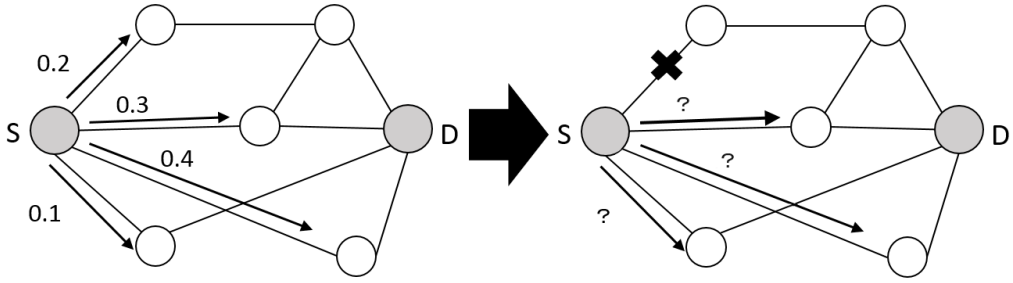

図 2.1: S-OSPF（左）と、リンク故障時の S-OSPF（右）

対策の一つとして、故障したリンクに隣接しているノードに於いてのみ分散を再計算するという方式が考案され、数理計算ソフトによるシミュレーションによってその性能はアプローチされてきた [9] が、実装難易度が高い為、実際のネットワークにその対策機能を実装する段階にまではまだ至っていない。

non-split S-OSPF

発ノードにおける分散処理をせず、その代わりに最も輻輳が少なくなるリンクを 1 つ選択してトラヒックを送信する non-split S-OSPF というルーチングがある。送られたトラヒックは中継ノードでは OSPF に従って着ノードまで送られていく。この様にして、ネットワーク全体での通信の負荷分散を図る。

ネットワーク輻輳を抑える性能は発ノードのトラヒック分散を行う S-OSPF より劣るが、分散比の計算を行う必要がないので、S-OSPF よりも実装が容易という利点がある。また、使用するネットワークの規模が大きくなる程、経路選択の自由度が高くなる為ネットワーク輻輳率を下げられる効果が期待できる。

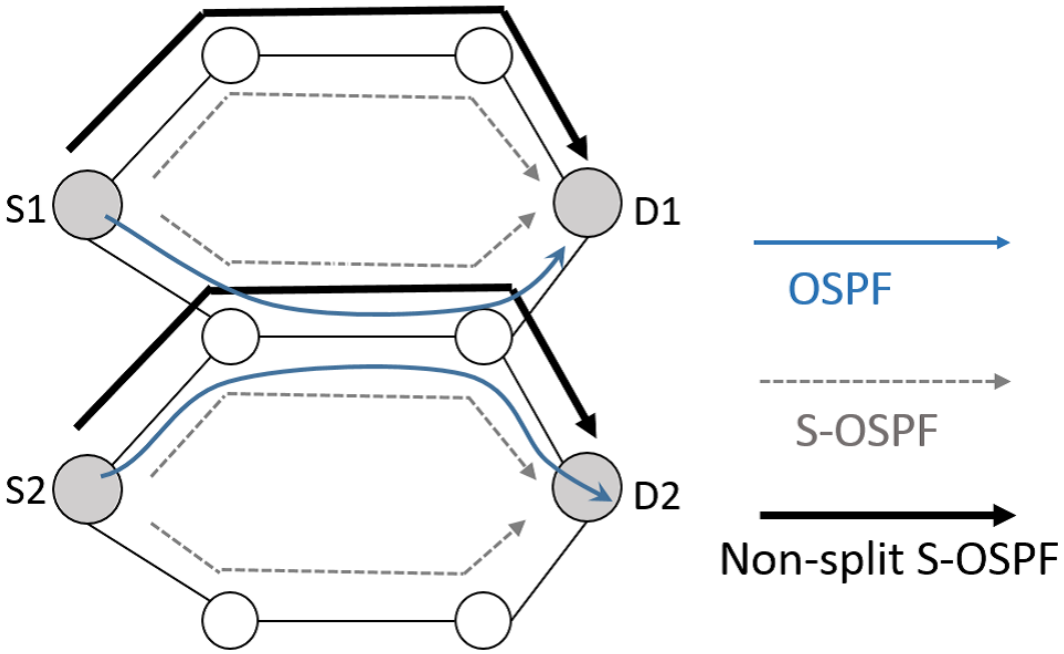

図 2.2: non-split S-OSPF と他のルーチング

2.2 OpenFlow

OpenFlow による S-OSPF 実装へのアプローチとその課題

Software-Defined Networking (以下,SDN) はネットワークの構造や構成、設定をソフトウェアによって柔軟かつ動的に変更する技術である。規格等がベンダーによって定められている既存の機器で構成されるネットワークに影響を与えずに SDN を実現するための標準として OpenFlow が用いられている。従来のネットワークを構成しているスイッチの設定や管理は一つずつ個別に行う必要があったのに対し、OpenFlow ではコントローラ部分をソフトウェア制御することによりネットワークの一元管理が可能となる。パケット処理のルールが記されたフローテーブルをコントローラがスイッチに書き込むことで管理は行われ、またテーブルは通信状況に応じて逐次更新される。

この OpenFlow によって、ネットワーク通信を行う際に発ノードから隣接ノードへ送るトラヒックの分配を適切に計算し発信する機能をルーターへ追加することで、既存のネットワークでは実現の難しい S-OSPF の分散機能を実装する取り組みがこれまでになされてきた。

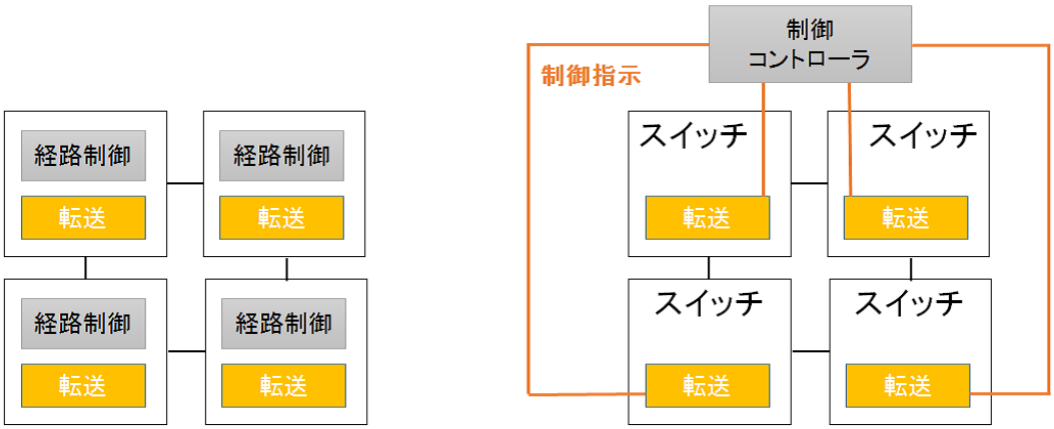

図 2.3: 従来のネットワーク（左）と OpenFlow によるネットワーク（右）

Mininet

実際に OpenFlow を動かすためのホストやスイッチの実機を用意することなく、一つの Linux カーネル上で複数の仮想ホストや仮想スイッチ、仮想ネットワークを手軽に自動で構成し、それらを組み合わせた任意の OpenFlow 環境を構築することが可能なツール。大規模なネットワークトポロジーも容易に作ったり変更したりすることができる。スタンフォード大学で開発されており、POX という Python で記述されたコントローラが用いられる。

第3章　実験環境

実験ネットワークの構築

実験環境のトポロジーを下図の様に構築した。下図に於いて h はホストを、s はスイッチをそれぞれ表す。

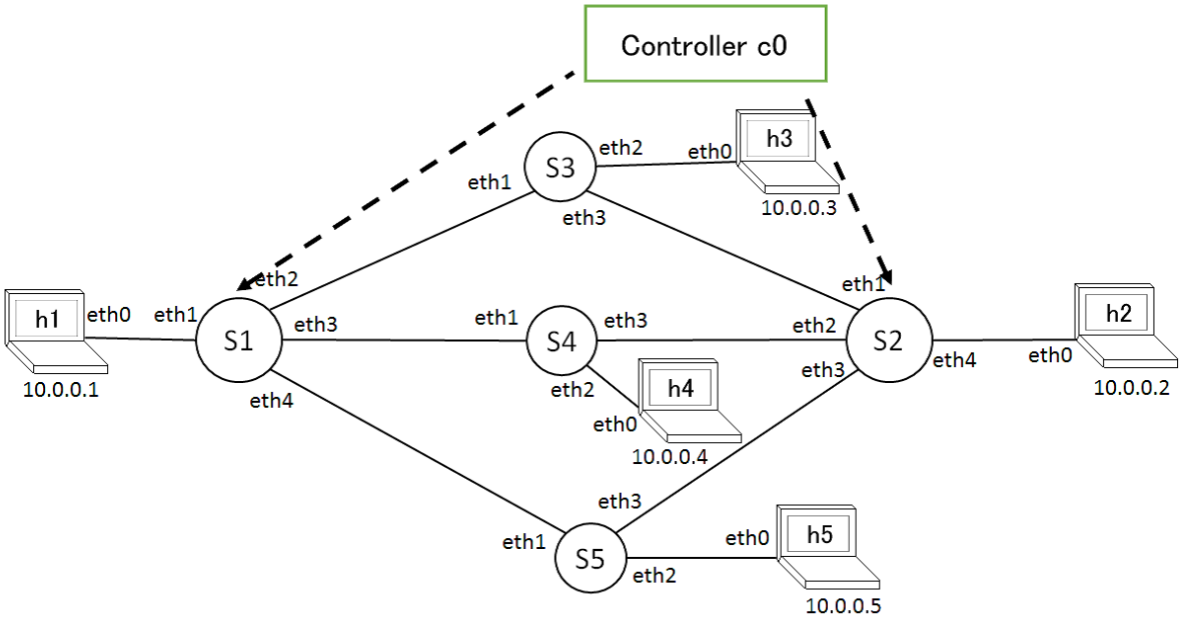

図 3.1: 実験構成

このトポロジーを構成しているホストの内、h1 と h2 をそれぞれ発ノード、着ノードとした。経路の分岐点にあたる s1 と s2 にネットワークの状態に応じてそれぞれのフローテーブルを書き換えて更新する指示をコントローラ c0 から出す。

フローテーブル

以下、通信経路を示す場合は通るスイッチの数字のみを順番に記するものとする（例えば s1s3s2 を通る経路の場合は 132 と記述する）。

初期状態では h1 から h2 へ送られたパケットは経路 132 を通るように、スイッチ s1 のフローテーブルには送信先が 10.0.0.2 のものは eth2 を、s2 のフローテーブルには送信先が 10.0.0.1 であるものは eth1 を通るようにそれぞれ記述されている。

また、この実験に用いるネットワークでは各ホストから送信されたパケットは OSPF によって最短経路を通るようにルーチングされると想定するものとし、スイッチ s3,s4,s5 のフローテーブルは送信宛の IP アドレスが 10.0.0.1 と 10.0.0.2 であるものはそれぞれ eth1,eth3 へ、他の中ノード宛のものは eth1 へ送るように予め書かれている。

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.731s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.2 actions=output:2
 cookie=0x0, duration=2.733s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.2 actions=output:2
 cookie=0x0, duration=2.732s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.731s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.1 actions=output:1
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.735s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.2 actions=output:4
 cookie=0x0, duration=2.736s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.2 actions=output:4
 cookie=0x0, duration=2.736s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.735s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.1 actions=output:1
*** s3 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.738s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.4 actions=output:1
 cookie=0x0, duration=2.738s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.5 actions=output:1
 cookie=0x0, duration=2.739s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.2 actions=output:3
 cookie=0x0, duration=2.74s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.2 actions=output:3
 cookie=0x0, duration=2.737s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.4 actions=output:1
 cookie=0x0, duration=2.74s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.739s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.737s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.5 actions=output:1
*** s4 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.74s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.5 actions=output:1
 cookie=0x0, duration=2.741s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.2 actions=output:3
 cookie=0x0, duration=2.742s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.2 actions=output:3
 cookie=0x0, duration=2.74s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.3 actions=output:1
 cookie=0x0, duration=2.742s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.741s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.739s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.3 actions=output:1
 cookie=0x0, duration=2.739s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.5 actions=output:1
*** s5 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.4 actions=output:1
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.2 actions=output:3
 cookie=0x0, duration=2.745s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.2 actions=output:3
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.3 actions=output:1
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.4 actions=output:1
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.1 actions=output:1
 cookie=0x0, duration=2.744s, table=0, n_packets=0, n_bytes=0, idle_age=2, ip,nw_dst=10.0.0.3 actions=output:1
```

図 3.2: 初期の全てのスイッチのフローテーブル

第4章　実装する機能と実験結果

使用中の経路に於いてトラヒックが混雑する可能性が生じた時、又は経路が切断された時には他の使用可能な経路を一つ選択し、そこで通信を行う様に切り替える機能を実装させる実験を行う。

4.1　平常時の通信分散

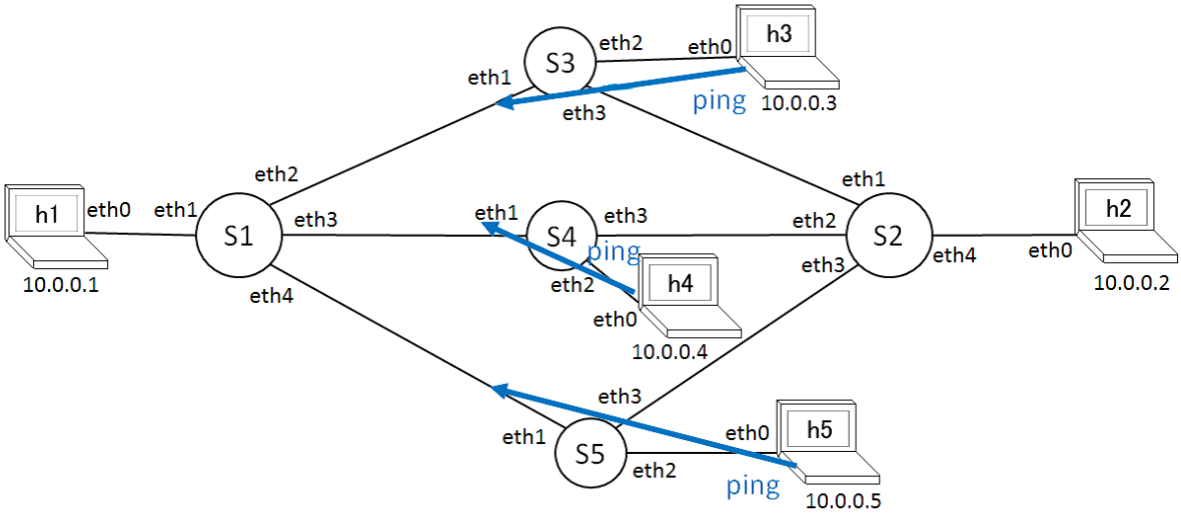

図 4.1: 中継ノードから ping

発着ノードである h1 と h2 の間で通信を行っている最中、h3,h4,h5 からパケットが飛ばされた際に、それぞれの通信に使われるリンクが重複するとそこで通信が混雑する可能性がある。そこで h1h2 間の通信で使用する経路を変更することで、ネットワーク全体で通信を分散させて混雑の回避を試みる。尚、この実験ではスイッチ同士を繋げているリンクのみを重複回避の対象として考慮するものとする。

例えば、h1h2 間の通信に経路 132 を使用している最中、h4 から h1 や h5 へパケットが飛ばされた場合は通信経路としてそれぞれ 41,415 が使われるため、スイッチ同士を繋げているリンクの中で重複するものが無いので何もしない。h4 から h3 へ飛ばされた場合は 413 が使われるので、リンク s1s3 が重複するのを避けるため、h1h2 間の通信に使用する経路を 152 へ切り替える。

中継ノードから発着ノードへパケット送信

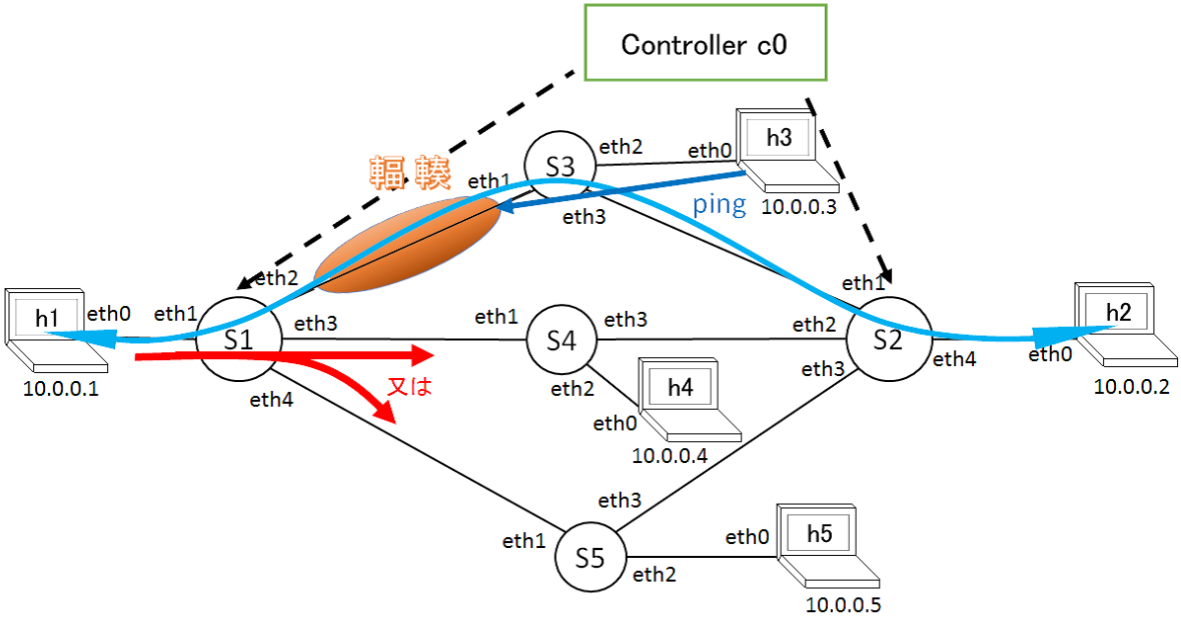

図 4.2: 経路 132 使用中に h3 から h1 へ ping

まずは使用中の経路を構成している中継ノードから発着ノードへパケットが飛ばされた場合の、輻輳回避の為の経路切り替え動作の実験を行う。

h1h2 間の通信に 132 を使用している最中に h3 から発ノード h1 へ ping を打つ。そうすると、その ping のパケットは経路 31 を通るため、このまま 132 を使用し続けているとリンク s1s3 が重複することになり、ここで輻輳を生む可能性がある。そこで、使用する経路を 142 又は 152 へランダムへ切り替える。h3 から ping が打たれたことを感知したコントローラ c0 が s1 と s2 に書かれたフローテーブルを書き換えて更新することで使用経路を切り替える。これにより経路の重複による輻輳の回避を試みる。フローテーブルの更新が正常に行われた場合、「OK Beer」という文字列と共に切り替え後の経路を表示させる。

結果

経路 132 を使用中、h3 から h1 へ ping を打った際に出力された表示と、表示後の s1 と s2 のフローテーブルは次ページの通りであった。s1 のフローテーブルには 10.0.0.2 宛のパケットは eth4 から送られるように、s2 のフローテーブルには 10.0.0.1 宛のパケットは eth3 を通るように記述されていた。これは h1h2 間の通信に 152 が使われる様になったことを意味する。また、表示とフローテーブルを確認後、実際に h1 から h2 へ飛ばす ping のパケットが経路 152 を通るかどうかを確認するために Wireshark を用いて s5-eth1 におけるパケットの動きをキャプチャした。

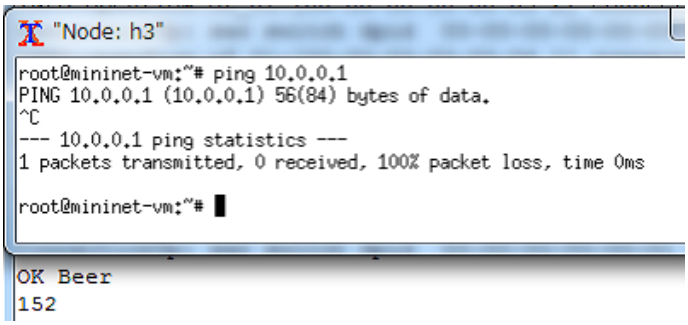

図 4.3: h3 から h1 へ ping を打った際の表示

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=24.002s, table=0, n_packets=0, n_bytes=0, idle_age=24, hard_age=12, ip,nw_dst=10.0.0.2 actions=output:4
 cookie=0x0, duration=24.002s, table=0, n_packets=0, n_bytes=0, idle_age=24, hard_age=12, arp,arp_tpa=10.0.0.2 actions=output:4
 cookie=0x0, duration=24.002s, table=0, n_packets=3, n_bytes=126, idle_age=12, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=24.002s, table=0, n_packets=0, n_bytes=0, idle_age=24, ip,nw_dst=10.0.0.1 actions=output:1
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=24.02s, table=0, n_packets=0, n_bytes=0, idle_age=24, ip,nw_dst=10.0.0.2 actions=output:4
 cookie=0x0, duration=24.02s, table=0, n_packets=0, n_bytes=0, idle_age=24, arp,arp_tpa=10.0.0.2 actions=output:4
 cookie=0x0, duration=24.02s, table=0, n_packets=0, n_bytes=0, idle_age=24, hard_age=12, arp,arp_tpa=10.0.0.1 actions=output:3
 cookie=0x0, duration=24.02s, table=0, n_packets=0, n_bytes=0, idle_age=24, hard_age=12, ip,nw_dst=10.0.0.1 actions=output:3
```

図 4.4: 経路変更後の s1 と s2 のフローテーブル

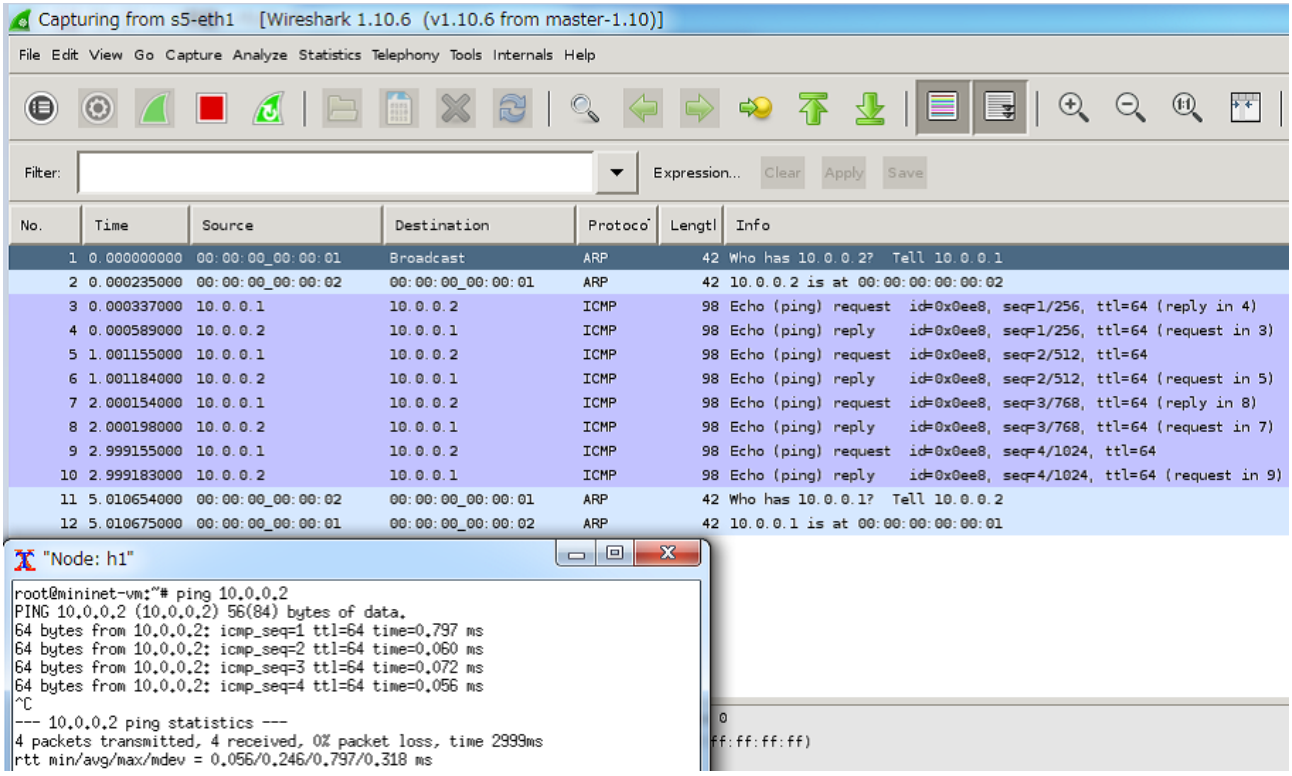

図 4.5: 経路変更後、h1 から h2 へ ping を打った際の s5-eth1 のパケットの動き

これらの結果により、h3 から h1 への ping により経路が 152 へ切り替えられたことが確認できた。

今度は h5 から再度 h1 へ ping を打った。同様の結果を得たことが確認できた。

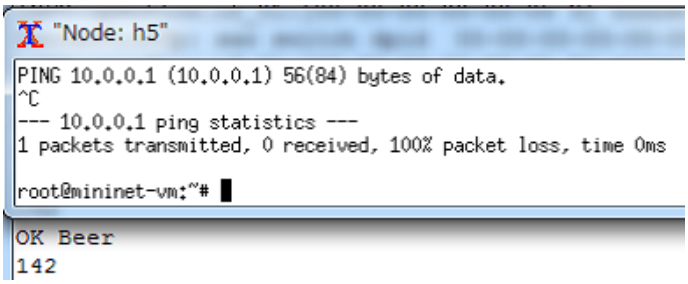

図 4.6: h5 から h1 へ ping を打った際の表示

```
mininet> dpctl dump-flows
*** s1 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=155.721s, table=0, n_packets=4, n_bytes=392, idle_age=102, hard_age=10, ip,nw_dst=10.0.0.2 actions=output:3
 cookie=0x0, duration=155.721s, table=0, n_packets=2, n_bytes=84, idle_age=100, hard_age=10, arp,arp_tpa=10.0.0.2 actions=output:3
 cookie=0x0, duration=155.721s, table=0, n_packets=8, n_bytes=336, idle_age=10, arp,arp_tpa=10.0.0.1 actions=output:1
 cookie=0x0, duration=155.721s, table=0, n_packets=4, n_bytes=392, idle_age=102, ip,nw_dst=10.0.0.1 actions=output:1
*** s2 ------------------------------------------------------------------------
NXST_FLOW reply (xid=0x4):
 cookie=0x0, duration=155.739s, table=0, n_packets=4, n_bytes=392, idle_age=102, ip,nw_dst=10.0.0.2 actions=output:4
 cookie=0x0, duration=155.739s, table=0, n_packets=2, n_bytes=84, idle_age=100, arp,arp_tpa=10.0.0.2 actions=output:4
 cookie=0x0, duration=155.739s, table=0, n_packets=2, n_bytes=84, idle_age=100, hard_age=10, arp,arp_tpa=10.0.0.1 actions=output:2
 cookie=0x0, duration=155.739s, table=0, n_packets=4, n_bytes=392, idle_age=102, hard_age=10, ip,nw_dst=10.0.0.1 actions=output:2
```

図 4.7: 経路変更後の s1 と s2 のフローテーブル

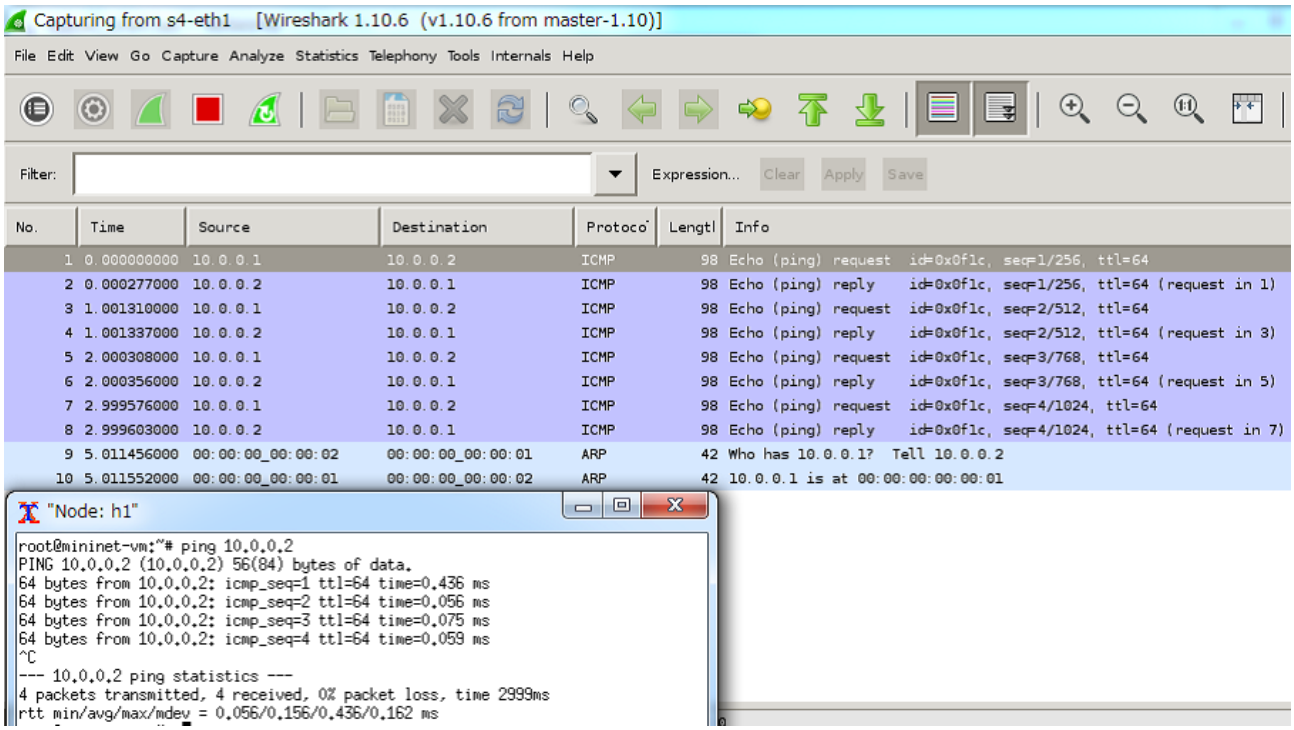

図 4.8: 経路変更後、h1 から h2 へ ping を打った際の s4-eth1 のパケットの動き

以下の結果の記述では、フローテーブルの表示と Wireshark によるパケットキャプチャの確認結果は省略する。

中継ノードから他の中継ノードへパケット送信

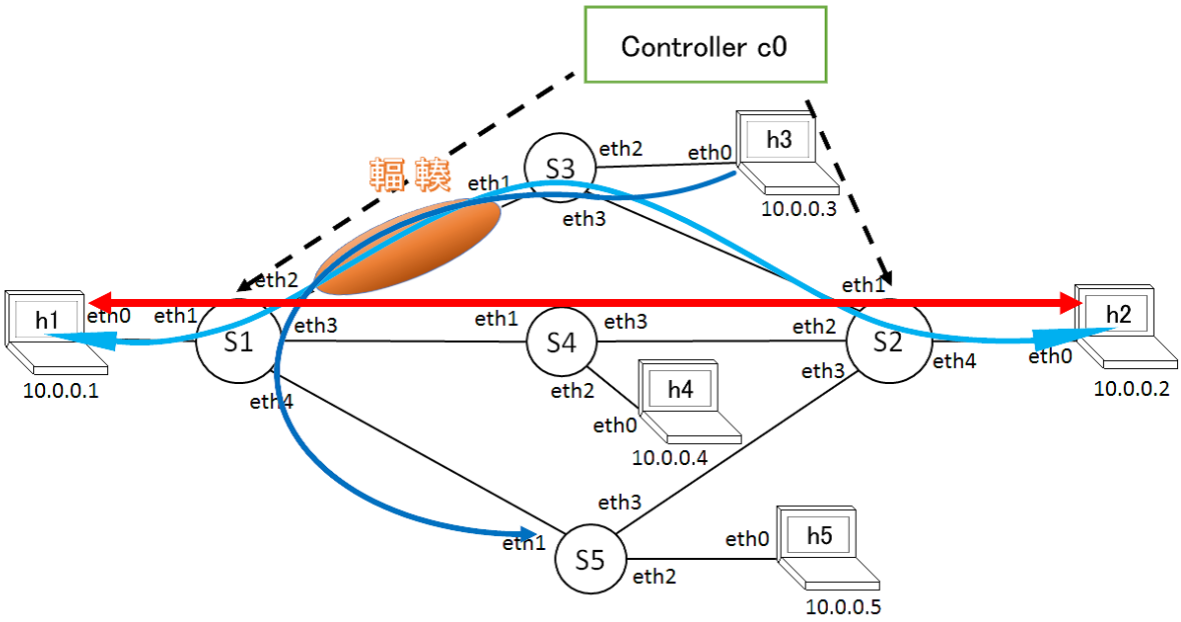

図 4.9: 経路 132 使用中に h3 から h5 へ ping

次に中継ノードから他の中継ノードへパケットが飛ばされた場合の、経路切り替え動作の実験を行う。

h1h2 間の通信に 132 を使用している最中に h3 から h5 へ ping を打つ。その ping のパケットは経路 315 を通るので、h5 へ届くまでにリンク s1s3 と s1s5 が使われる為、唯一重複するリンクがない経路 142 へ自動で切り替えるように s1 と s2 のフローテーブルを書き換えて更新する。フローテーブルの更新が正常に行われた場合、「OK Beer」という文字列と共に切り替え後の経路 142 を表示させる。

結果

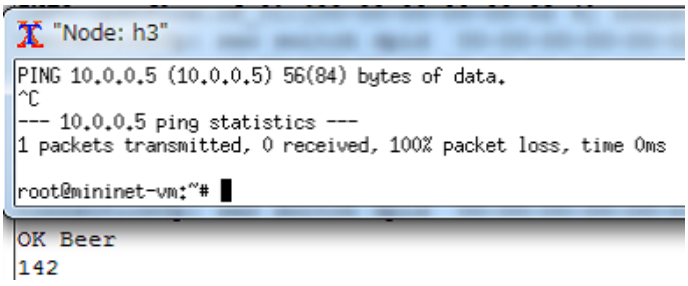

図 4.10: h3 から h5 へ ping を打った際の経路変更

文字列 OK Beer と共に変更後の経路 142 が表示されたことが確認できた。

4.2　経路切断時の対応動作

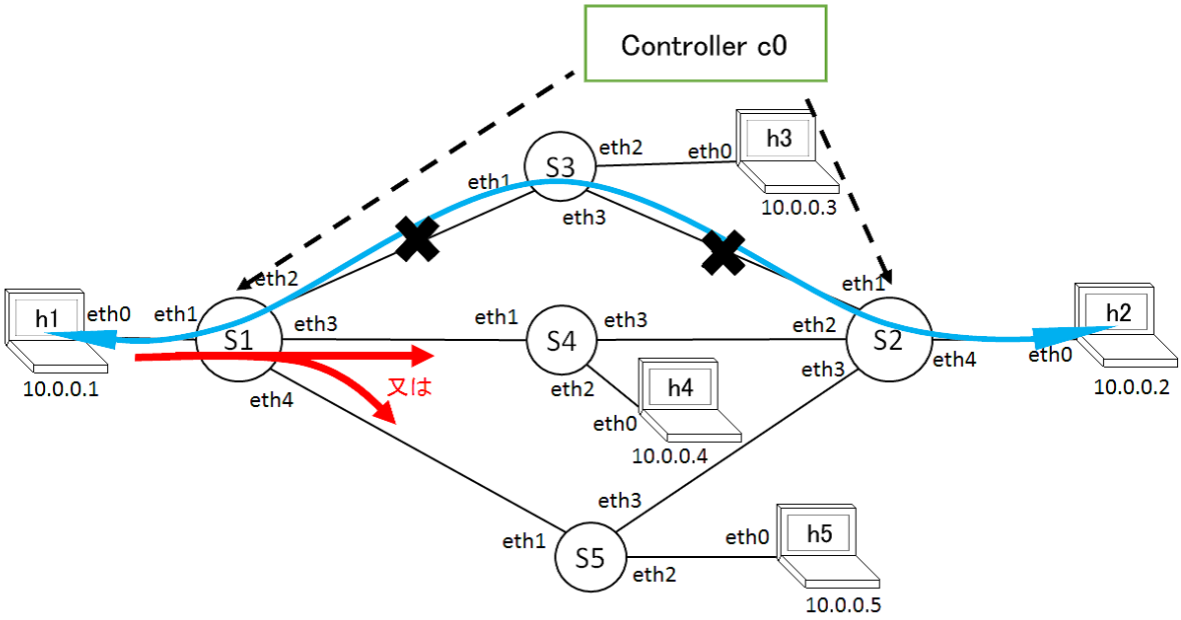

図 4.11: 経路 132 使用中にリンク s1s3 又は s2s3 を切断

今度は通信中に使用中の経路を構成しているリンクが切断された際の対応動作の実験を行う。

h1h2 間の通信に 132 を使用している最中にリンク s1s3 又は s2s3 を切断し、経路 132 を使用不能な状態にする。この時リンクの切断を感知したコントローラー c0 が、残りの使用可能な経路 142 か 152 の何れかをランダムで選択して自動で切り替えるようにスイッチ s1s2 のフローテーブルを書き換えて更新することで、リンク切断後も h1h2 間の通信を可能な状態に保つ。リンク切断及びフローテーブルの更新後、切断されたリンクの場所、全てのスイッチ間同士を繋いでいるリンクの状態の一覧及び切り替え後の経路を表示させる。

また、更に切り替えた後の経路もリンクが切断されて使用不能になった場合、残りの一つの経路へ自動で切り替える。この対応動作の実装により、最大二回までの経路切断へ耐えうる通信ネットワークを築くことを目指す。

結果

まず、経路 132 の使用中にリンク s1s3 を切断した際の表示を以下に示す。dpid は切断されたリンクに隣接しているスイッチの番号、port はスイッチの eth 番号を、またリンク状態一覧において、0 はリンクが正常な状態を、1 は切断されている状態をそれぞれ表す。

```
PortStatus has been changed
switch dpid 1 port 2 status = down config = 1
s1s3: 1 s2s3: 0
s1s4: 0 s2s4: 0
s1s5: 0 s2s5: 0
142
PortStatus has been changed
switch dpid 3 port 1 status = down config = 1
s1s3: 1 s2s3: 0
s1s4: 0 s2s4: 0
s1s5: 0 s2s5: 0
142
```

図 4.12: リンク s1s3 を切断した際の際の全てリンクの状態と変更後の経路

これにより、リンク切断後経路が 142 へ自動で切り替えられたことが確認できた。次に、選択された経路 142 を構成しているリンク s1s4 を切断した。

```
PortStatus has been changed
switch dpid 4 port 1 status = down config = 1
s1s3: 1 s2s3: 0
s1s4: 1 s2s4: 0
s1s5: 0 s2s5: 0
152
PortStatus has been changed
switch dpid 1 port 3 status = down config = 1
s1s3: 1 s2s3: 0
s1s4: 1 s2s4: 0
s1s5: 0 s2s5: 0
152
```

図 4.13: リンク s1s4 を切断した際の際の全てリンクの状態と変更後の経路

結果、残りの使用可能な経路である 152 へ切り替えられたことが確認できた。

4.3　経路切断時の通信分散

通信分散と経路破損時の対応動作の応用を試みる。一部の経路が切断されて使用不能である状態で中継ノードからパケットが飛ばされて経路変更が行われる場合、ここでリンクが切断されているものを選択してしまうと、h1h2 間の通信が出来なくなる為、それを避けるために必ず使用できるものを選択するようにする。

経路切断中に中継ノードから発着ノードへパケット送信

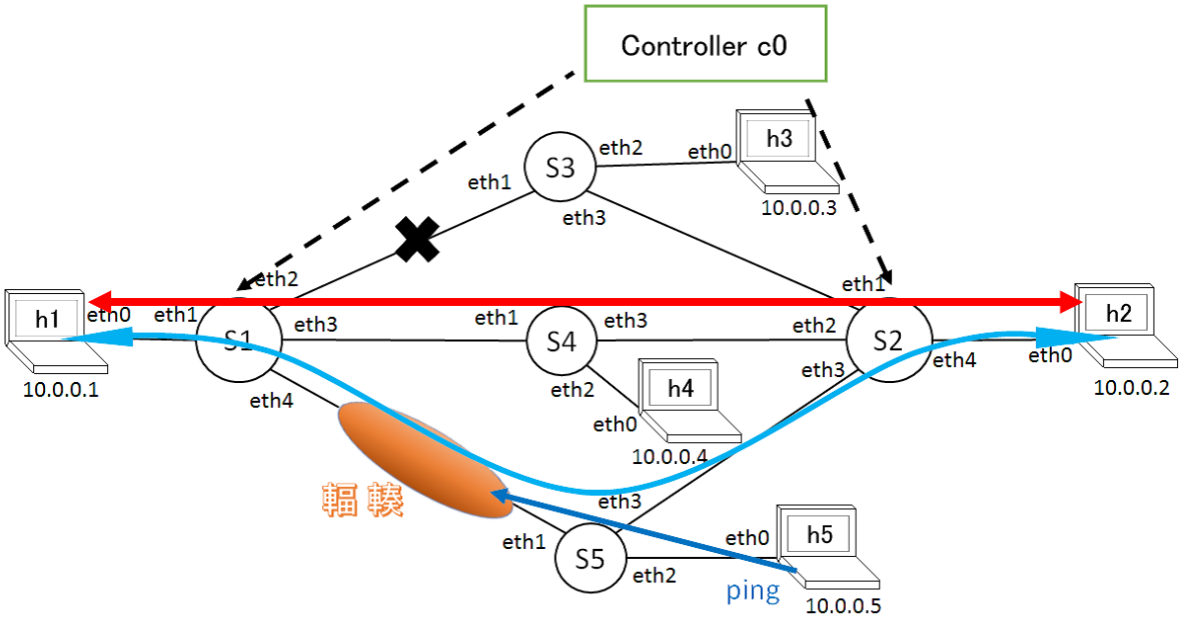

図 4.14: 経路 152 使用中且つ s1s3 が切断された状態で h5 から h1 へ ping

リンク s1s3 を切断し、使用経路が 152 へ切り替えられた後、h5 から h1 へ ping を打つ。ここで切り替える経路として 132 を選択してしまうと、132 は使用不能のままなので h1h2 間の通信が出来なくなってしまう。その為必ず使用可能である経路 142 を選択させる。

結果

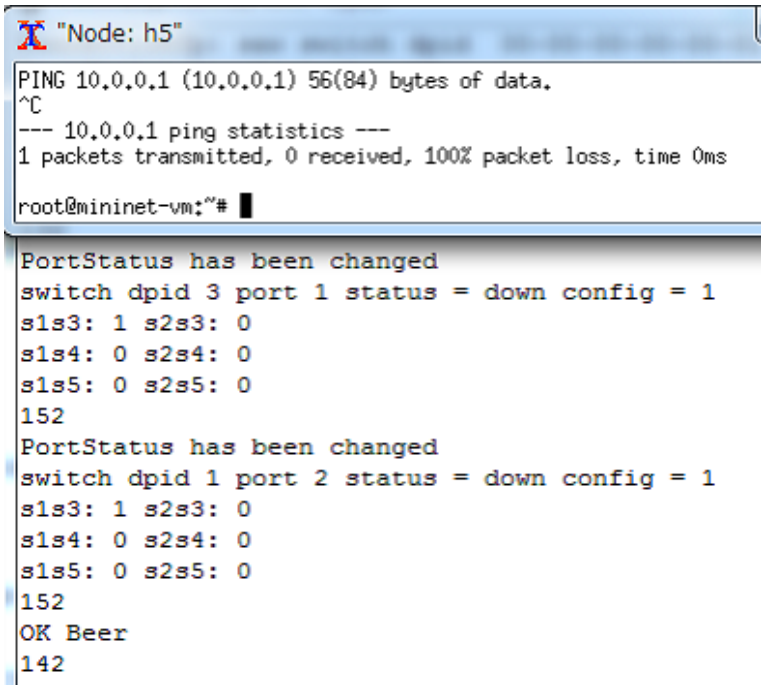

図 4.15: リンク s1s3 が切断された状態で経路 152 を使用中に h5 から h1 へ ping を打った際の表示

リンク s1s3 が切断されている状態で且つ経路 152 を使用している最中に h5 から h1 へ ping を打った結果、経路 142 が選択された。これにより、誤って 132 を選択して h1h2 間の通信が不能になるのを避けたことが確認できた。

経路切断中に中継ノードから他の中継ノードへパケット送信

直後に次に h4 から h5 へ ping を打つと、本来選択されるはずの経路 132 は使用不能のままなので、経路は変更されず、文字列 error が出る。

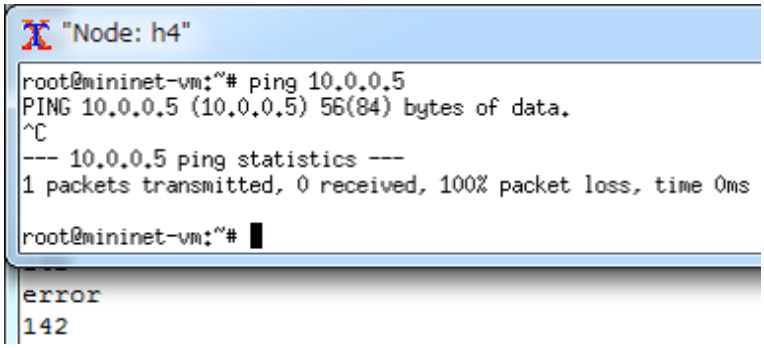

図 4.16: リンク s1s3 が切断された状態で経路 142 を使用中に h4 から h5 へ ping を打った際の表示

第5章　まとめ

これらの一連の実験の後、再度使用中の経路に繋がれたホストから他のホストへ ping を打つ実験を、リンクの切断と再接続を繰り返しながら複数回行った。結果、何れの実験も中継ノードから ping のパケットが飛ばされる際に使われるリンクと重複せず、且つリンクが切断されていない使用可能な経路が存在する場合、その経路を代替経路として選択したことを示した。存在しない場合前述通り error と表示され経路の変更はなされなかったが、仮に経路を変更しないことでリンクが重複したことによって通信混雑が発生しても、誤って使用不能の経路を選択して通信不能になるよりはリスクが少ないと判断し、その様に制御した。

以上により、コントローラに自動で経路を切り替えさせる機能を実装することにより、

- 通信中に中継ノードから他のノードへパケットが飛ばされた際、その送信に使われる経路のリンクと、h1h2 間の通信に使われる経路のリンクの重複を可能な限り避けることで、ネットワーク全体の通信を分散させる
- 使用中の経路が切断された際に他の使用可能な経路を自動で選択し切り替えることで、h1h2 間の通信が不能になるのを防ぐ

上二つの機能を実現できたことが確認できた。

謝　　辞

本研究を進めるにあたり、ご指導を頂いた大木英司教授、Nattapong Kitsuwan 教授に心より感謝を申し上げます。また、研究を進めるにあたり相談に乗って頂いた内藤郁之先輩、仲程康憲先輩、研究を参考にさせて頂いた本間奨先輩、角田俊一先輩、またお世話になった同輩、後輩の方々に厚く感謝を申し上げます。

参考文献

[1] K. Ishizu, H. Murakami, and H. Harada, “ Cognitive wireless network infrastructure and restoration activities for the earthquake disaster” Personal Multimedia Communications (WPMC), 2011 14th International Symposium on, pp. 1-5, 2011

[2] A.K. Mishra and A. Sahoo, ”S-OSPF: a traffic engineeringsolution for OSPF based on best effrot networks” IEEE Globecom 2007, pp. 1845-1849, 2007.

[3] http://www.ilog.com/,2014.

[4] M. Honma, S. Tsunoda, and E. Oki, ”Load-balanced shortest-path-based routing with even traffic splitting,” Communications (APCC), 2012 18th Asia-Pacific Conference on, pp.361-365, October 2012.

[5] N. McKeown, T. Anderson, H. Balakrishnan, G.Parulkar, L. Peterson, J. Rexford, S. Shenker,and J.Turner, “ Openflow: enabling innovation in campus networks, ” SIGCOMM Comput. Commun.Rev.,38(2):69-74, 2008.

[6] Y Nakahodo, T. Naito, and E. Oki, ”Implementation of smart-OSPF in hybrid software-defined network” IEEE Network Infrastructure and Digital Content (ICNIDC), 2014 4th IEEE International Conference on, pp.374 - 378, 2014.

[7] http://mininet.org/

[8] S. Tsunoda, A.H.A. Muktadir, and E. Oki,”Load-Balanced Non-Split Shortest-Path-Based Routing with Hose Model and Its Demonstration”,IEICE Trans. Commun.,E96-B/ 5, 1130-1140,2013

[9] 本間奨, 大木英司, “リンク故障を考慮した負荷分散 IP ルーチング方式,” 電子情報通信学会論文誌 B, vol. J97-B, no. 11, pp. 1085-1095, Nov. 2014.

研究実績

外部発表

鈴木龍太, 大木英司, “OpenFlow による高信頼・トラヒック分散ネットワークの構築,” 電子情報通信学会東京支部学生会第 21 回研究発表会, no. 133, Mar. 2016.

付録

ソースコード

トポロジー設定ファイル

```
from mininet.topo import Topo

class MyTopo( Topo ):
    "Simple topology example."

    def __init__( self ):
        "Create custom topo."

        # Initialize topology
        Topo.__init__( self )

        # Add hosts and switches
        leftHost = self.addHost( 'h1' , ip='10.0.0.1' , mac='00:00:00:00:00:01' )
        rightHost = self.addHost( 'h2' , ip='10.0.0.2' , mac='00:00:00:00:00:02' )
        centerHost3 = self.addHost( 'h3' , ip='10.0.0.3' , mac='00:00:00:00:00:03' )
        centerHost4 = self.addHost( 'h4' , ip='10.0.0.4' , mac='00:00:00:00:00:04' )
        centerHost5 = self.addHost( 'h5' , ip='10.0.0.5' , mac='00:00:00:00:00:05' )

        leftSwitch = self.addSwitch( 's1' , ip='10.0.0.11' , dpid='0000000000000001' )
        rightSwitch = self.addSwitch( 's2' , ip='10.0.0.12' , dpid='0000000000000002' )
        centerSwitch3 = self.addSwitch( 's3' , ip='10.0.0.13' , dpid='0000000000000003' )
        centerSwitch4 = self.addSwitch( 's4' , ip='10.0.0.14' , dpid='0000000000000004' )
        centerSwitch5 = self.addSwitch( 's5' , ip='10.0.0.15' , dpid='0000000000000005' )

        # Add links
        self.addLink( leftHost, leftSwitch )
        self.addLink( leftSwitch, centerSwitch3 )
        self.addLink( leftSwitch, centerSwitch4 )
        self.addLink( leftSwitch, centerSwitch5 )
        self.addLink( centerSwitch3, centerHost3 )
        self.addLink( centerSwitch4, centerHost4 )
        self.addLink( centerSwitch5, centerHost5 )
        self.addLink( centerSwitch3, rightSwitch )
        self.addLink( centerSwitch4, rightSwitch )
        self.addLink( centerSwitch5, rightSwitch )
        self.addLink( rightSwitch, rightHost )


topos = { 'mytopo': ( lambda: MyTopo() ) }
```

ルーチング設定ファイル

```
from pox.core import core
import pox.openflow.libopenflow_01 as of
from pox.lib.util import dpidToStr
from pox.lib.addresses import EthAddr
from pox.lib.addresses import IPAddr
from pox.openflow.of_json import *
import random

s1s3 = 0
s2s3 = 0
s1s4 = 0
s2s4 = 0
s1s5 = 0
s2s5 = 0
nowroot = 132
out_port1 = 2
out_port2 = 1
log = core.getLogger()

def launch ():
  core.openflow.addListenerByName("PortStatus", _handle_PortStatus)
  core.openflow.addListenerByName("ConnectionUp", _handle_ConnectionUp)
  core.openflow.addListenerByName("PacketIn", _handle_PacketIn)
  log.info("Switch running.")


def _handle_PortStatus (event):
  global s1s3
  global s2s3
  global s1s4
  global s2s4
  global s1s5
  global s2s5
  global nowroot
  global out_port1
  global out_port2
  print "PortStatus has been changed"
  primary_down_flag = 0
  port_config = event.ofp.desc.config
  if event.ofp.desc.config:
    port_status = "down"
  else:
    port_status = "up"
  print "switch dpid", event.dpid, "port", event.port,
  print "status = " + port_status + " config = "+str(event.ofp.desc.config)

#s1--/--s3
  if ((event.dpid == 1 and event.port == 2) \
      or (event.dpid == 3 and event.port == 1)) \
    and event.ofp.desc.config:
    s1s3 = 1
    if (nowroot == 142 or nowroot == 152):
      pass
    else:
      if s1s4 == 1 or s2s4 == 1:
        out_port1 = 4
      elif s1s5 == 1 or s2s5 == 1:
        out_port1 = 3
      else:
        out_port1 = random.randint(3,4)

      if out_port1 == 3:
        out_port2 = 2
        nowroot = 142
      else:
        out_port2 = 3
        nowroot = 152
```

```
68
69    #s3--/--s2
70      elif ((event.dpid == 3 and event.port == 3) \
71            or (event.dpid == 2 and event.port == 1)) \
72        and event.ofp.desc.config:
73        s2s3 = 1
74        if (nowroot == 142 or nowroot == 152):
75          pass
76        else:
77          if s1s4 == 1 or s2s4 == 1:
78            out_port1 = 4
79          elif s1s5 == 1 or s2s5 == 1:
80            out_port1 = 3
81          else:
82            out_port1 = random.randint(3,4)
83
84          if out_port1 == 3:
85            out_port2 = 2
86            nowroot = 142
87          else:
88            out_port2 = 3
89            nowroot = 152
90
91
92    #s1-----s3
93      elif ((event.dpid == 1 and event.port == 2) \
94            or (event.dpid == 3 and event.port == 1)) \
95        and str(event.ofp.desc.config) != 1:
96       s1s3 = 0
97
98    #s3-----s2
99      elif ((event.dpid == 3 and event.port == 3) \
100           or (event.dpid == 2 and event.port == 1)) \
101       and str(event.ofp.desc.config) != 1:
102      s2s3 = 0
103
104
105   #s1--/--s4
106     elif ((event.dpid == 1 and event.port == 3) \
107           or (event.dpid == 4 and event.port == 1)) \
108       and event.ofp.desc.config:
109       s1s4 = 1
110       if (nowroot == 132 or nowroot == 152):
111         pass
112       else:
113         if s1s3 == 1 or s2s3 == 1:
114           out_port1 = 4
115         elif s1s5 == 1 or s2s5 == 1:
116           out_port1 = 2
117         else:
118           n = random.randint(1,2)
119           if n == 1:
120             out_port1 = 2
121           else:
122             out_port1 = 4
123
124         if out_port1 == 2:
125           out_port2 = 1
126           nowroot = 132
127         else:
128           out_port2 = 3
129           nowroot = 152
130
131   #s4--/--s2
132     elif ((event.dpid == 4 and event.port == 3) \
133           or (event.dpid == 2 and event.port == 2)) \
134       and event.ofp.desc.config:
135       s2s4 = 1
136       if (nowroot == 132 or nowroot == 152):
137         pass
138       else:
139         if s1s3 == 1 or s2s3 == 1:
```

```
          out_port1 = 4
        elif s1s5 == 1 or s2s5 == 1:
          out_port1 = 2
        else:
          n = random.randint(1,2)
          if n == 1:
            out_port1 = 2
          else:
            out_port1 = 4

        if out_port1 == 2:
          out_port2 = 1
          nowroot = 132
        else:
          out_port2 = 3
          nowroot = 152


    #s1-----s4
      elif ((event.dpid == 1 and event.port == 3) \
            or (event.dpid == 4 and event.port == 1)) \
       and str(event.ofp.desc.config) != 1:
       s1s4 = 0

    #s4-----s2
      elif ((event.dpid == 4 and event.port == 3) \
            or (event.dpid == 2 and event.port == 2)) \
       and str(event.ofp.desc.config) != 1:
       s2s4 = 0


    #s1--/--s5
      elif ((event.dpid == 1 and event.port == 4) \
            or (event.dpid == 5 and event.port == 1)) \
       and event.ofp.desc.config:
       s1s5 = 1
       if (nowroot == 132 or nowroot == 142):
         pass
       else:
         if s1s3 == 1 or s2s3 == 1:
           out_port1 = 3
         elif s1s4 == 1 or s2s4 == 1:
           out_port1 = 2
         else:
           out_port1 = random.randint(2,3)

         if out_port1 == 2:
           out_port2 = 1
           nowroot = 132
         else:
           out_port2 = 2
           nowroot = 142

    #s5--/--s2
      elif ((event.dpid == 5 and event.port == 3) \
            or (event.dpid == 2 and event.port == 3)) \
       and event.ofp.desc.config:
       s2s5 = 1
       if (nowroot == 132 or nowroot == 142):
         pass
       else:
         if s1s3 == 1 or s2s3 == 1:
           out_port1 = 3
         elif s1s4 == 1 or s2s4 == 1:
           out_port1 = 2
         else:
           out_port1 = random.randint(2,3)

         if out_port1 == 2:
           out_port2 = 1
           nowroot = 132
         else:
```

```
          out_port2 = 2
          nowroot = 142


  #s1-----s5
    elif ((event.dpid == 1 and event.port == 4) \
          or (event.dpid == 5 and event.port == 1)) \
      and str(event.ofp.desc.config) != 1:
     s1s5 = 0

  #s5-----s2
    elif ((event.dpid == 5 and event.port == 3) \
          or (event.dpid == 2 and event.port == 3)) \
      and str(event.ofp.desc.config) != 1:
     s2s5 = 0


    mod_flow(1,'arp',IPAddr("10.0.0.2"),out_port1)
    mod_flow(2,'arp',IPAddr("10.0.0.1"),out_port2)
    mod_flow(1,'ip',IPAddr("10.0.0.2"),out_port1)
    mod_flow(2,'ip',IPAddr("10.0.0.1"),out_port2)
    print "s1s3:",s1s3, "s2s3:",s2s3
    print "s1s4:",s1s4, "s2s4:",s2s4
    print "s1s5:",s1s5, "s2s5:",s2s5
    print nowroot


  def _handle_PacketIn (event):
    m = { 'buffer_id' : event.ofp.buffer_id,
          'total_len' : event.ofp._total_len,
          'in_port' : event.ofp.in_port,
          'reason' : event.ofp.reason,
          #'data' : event.data,
        }
    m['payload'] = fix_parsed(event.parsed)
    global s1s3
    global s2s3
    global s1s4
    global s2s4
    global s1s5
    global s2s5
    global nowroot
    global out_port1
    global out_port2

  #s3 -> s1,s2
    if m['payload']['payload']['protodst']=="10.0.0.3" \
       and (m['payload']['payload']['protosrc']=="10.0.0.1" \
            or m['payload']['payload']['protosrc']=="10.0.0.2"):
      if nowroot != 132:
        pass
      elif (s1s4 == 1 or s2s4 == 1) and (s1s5 == 1 or s2s5 == 1):
        print "error"
      elif s1s4 == 1 or s2s4 == 1:
        out_port1 = 4
        out_port2 = 3
        nowroot = 152
        print "OK Beer"
      elif s1s5 == 1 or s2s5 == 1:
        out_port1 = 3
        out_port2 = 2
        nowroot = 142
        print "OK Beer"
      else:
        n = random.randint(1,2)
        if n == 1:
          out_port1 = 3
          out_port2 = 2
          nowroot = 142
```

```
281           print "OK Beer"
282         else:
283           out_port1 = 4
284           out_port2 = 3
285           nowroot = 152
286           print "OK Beer"
287
288   #s4 -> s1,s2
289     elif m['payload']['payload']['protodst']=="10.0.0.4" \
290          and (m['payload']['payload']['protosrc']=="10.0.0.1" \
291               or m['payload']['payload']['protosrc']=="10.0.0.2"):
292       if nowroot != 142:
293         pass
294       elif (s1s3 == 1 or s2s3 == 1) and (s1s5 == 1 or s2s5 == 1):
295         print "error"
296       elif s1s3 == 1 or s2s3 == 1:
297         out_port1 = 4
298         out_port2 = 3
299         nowroot = 152
300         print "OK Beer"
301       elif s1s5 == 1 or s2s5 == 1:
302         out_port1 = 2
303         out_port2 = 1
304         nowroot = 132
305         print "OK Beer"
306       else:
307         n = random.randint(1,2)
308         if n == 1:
309           out_port1 = 2
310           out_port2 = 1
311           nowroot = 132
312           print "OK Beer"
313         else:
314           out_port1 = 4
315           out_port2 = 3
316           nowroot = 152
317           print "OK Beer"
318
319   #s5 -> s1,s2
320     elif m['payload']['payload']['protodst']=="10.0.0.5" \
321          and (m['payload']['payload']['protosrc']=="10.0.0.1" \
322               or m['payload']['payload']['protosrc']=="10.0.0.2"):
323       if nowroot != 152:
324         pass
325       elif (s1s3 == 1 or s2s3 == 1) and (s1s4 == 1 or s2s4 == 1):
326         print "error"
327       elif s1s3 == 1 or s2s3 == 1:
328         out_port1 = 3
329         out_port2 = 2
330         nowroot = 142
331         print "OK Beer"
332       elif s1s4 == 1 or s2s4 == 1:
333         out_port1 = 2
334         out_port2 = 1
335         nowroot = 132
336         print "OK Beer"
337       else:
338         n = random.randint(1,2)
339         if n == 1:
340           out_port1 = 2
341           out_port2 = 1
342           nowroot = 132
343           print "OK Beer"
344         else:
345           out_port1 = 3
346           out_port2 = 2
347           nowroot = 142
348           print "OK Beer"
349
350
```

```
#s3 <-> s4
  elif (m['payload']['payload']['protosrc']=="10.0.0.3" \
        and m['payload']['payload']['protodst']=="10.0.0.4") \
    or (m['payload']['payload']['protosrc']=="10.0.0.4" \
        and m['payload']['payload']['protodst']=="10.0.0.3"):
    if s1s5 == 1 or s2s5 == 1:
      print "error"
    elif nowroot == 152:
      pass
    else:
      print "OK Beer"
      out_port1 = 4
      out_port2 = 3
      nowroot = 152

#s3 <-> s5
  elif (m['payload']['payload']['protosrc']=="10.0.0.3" \
        and m['payload']['payload']['protodst']=="10.0.0.5") \
    or (m['payload']['payload']['protosrc']=="10.0.0.5" \
        and m['payload']['payload']['protodst']=="10.0.0.3"):
    if s1s4 == 1 or s2s4 == 1:
      print "error"
    elif nowroot == 142:
      pass
    else:
      print "OK Beer"
      out_port1 = 3
      out_port2 = 2
      nowroot = 142

#s4 <-> s5
  elif (m['payload']['payload']['protosrc']=="10.0.0.4" \
        and m['payload']['payload']['protodst']=="10.0.0.5") \
    or (m['payload']['payload']['protosrc']=="10.0.0.5" \
        and m['payload']['payload']['protodst']=="10.0.0.4"):
    if s1s3 == 1 or s2s3 == 1:
      print "error"
    elif nowroot == 132
      pass
    else:
      print "OK Beer"
      out_port1 = 2
      out_port2 = 1
      nowroot = 132

  mod_flow(1,'arp',IPAddr("10.0.0.2"),out_port1)
  mod_flow(2,'arp',IPAddr("10.0.0.1"),out_port2)
  mod_flow(1,'ip',IPAddr("10.0.0.2"),out_port1)
  mod_flow(2,'ip',IPAddr("10.0.0.1"),out_port2)
  print nowroot

def _handle_ConnectionUp (event):
  dpid=dpidToStr(event.dpid)
  print "ConnectionUp: saw switch dpid ", dpid , event.dpid
  m = random.randint(1,2)

  add_flow(1,'arp',IPAddr("10.0.0.2"),2)
  add_flow(1,'arp',IPAddr("10.0.0.1"),1)
  add_flow(1,'ip',IPAddr("10.0.0.2"),2)
  add_flow(1,'ip',IPAddr("10.0.0.1"),1)

  add_flow(2,'arp',IPAddr("10.0.0.2"),4)
  add_flow(2,'arp',IPAddr("10.0.0.1"),1)
  add_flow(2,'ip',IPAddr("10.0.0.2"),4)
  add_flow(2,'ip',IPAddr("10.0.0.1"),1)

  add_flow(3,'arp',IPAddr("10.0.0.2"),3)
  add_flow(3,'arp',IPAddr("10.0.0.1"),1)
  add_flow(3,'ip',IPAddr("10.0.0.2"),3)
```

```
420     add_flow(3,'ip',IPAddr("10.0.0.1"),1)
421     add_flow(3,'arp',IPAddr("10.0.0.4"),1)
422     add_flow(3,'arp',IPAddr("10.0.0.5"),1)
423     add_flow(3,'ip',IPAddr("10.0.0.4"),1)
424     add_flow(3,'ip',IPAddr("10.0.0.5"),1)
425
426     add_flow(4,'arp',IPAddr("10.0.0.2"),3)
427     add_flow(4,'arp',IPAddr("10.0.0.1"),1)
428     add_flow(4,'ip',IPAddr("10.0.0.2"),3)
429     add_flow(4,'ip',IPAddr("10.0.0.1"),1)
430     add_flow(4,'arp',IPAddr("10.0.0.3"),1)
431     add_flow(4,'arp',IPAddr("10.0.0.5"),1)
432     add_flow(4,'ip',IPAddr("10.0.0.3"),1)
433     add_flow(4,'ip',IPAddr("10.0.0.5"),1)
434
435     add_flow(5,'arp',IPAddr("10.0.0.2"),3)
436     add_flow(5,'arp',IPAddr("10.0.0.1"),1)
437     add_flow(5,'ip',IPAddr("10.0.0.2"),3)
438     add_flow(5,'ip',IPAddr("10.0.0.1"),1)
439     add_flow(5,'arp',IPAddr("10.0.0.3"),1)
440     add_flow(5,'arp',IPAddr("10.0.0.4"),1)
441     add_flow(5,'ip',IPAddr("10.0.0.3"),1)
442     add_flow(5,'ip',IPAddr("10.0.0.4"),1)
443
444
445   def add_flow (dpid, proto, nw_dst, out_port):
446     msg = of.ofp_flow_mod(command=of.OFPFC_ADD)
447     if proto=='arp':
448       msg.match.dl_type = 0x0806
449     elif proto=='ip':
450       msg.match.dl_type = 0x0800
451     msg.match.nw_dst = IPAddr(nw_dst)
452     msg.actions.append(of.ofp_action_output(port = out_port))
453     core.openflow.sendToDPID(dpid, msg)
454
455   def mod_flow (dpid, proto, nw_dst, out_port):
456     msg = of.ofp_flow_mod(command=of.OFPFC_MODIFY_STRICT)
457     if proto=='arp':
458       msg.match.dl_type = 0x0806
459     elif proto=='ip':
460       msg.match.dl_type = 0x0800
461     msg.match.nw_dst = IPAddr(nw_dst)
462     msg.actions.append(of.ofp_action_output(port = out_port))
463     core.openflow.sendToDPID(dpid, msg)
464
465
```